Programming Floppy Disk Controllers

[Home]

[Programming]

The purpose of this document is to provide information related to programming the NEC µPD765 and the Intel 82072/7 Floppy Disk Controllers (FDCs), by use of the registers. The document is split into several sections:

Overview

The PC usually uses the NEC µPD765 floppy disk controller. The AT can also incorporate an Intel 82072A controller, while the PS/2 uses an Intel 82077A. The µPD765 and the ROM code in the controller form a microcontroller, and this handles the majority of the work of the controller. All PC compatibles have FDCs which are compatible with the controllers described in this document.

This document describes the registers used to interface with the controller, and the commands which it recognises.

There are a lot of delays involved in communicating with the controller. These delays are for a variety of reasons, including the time needed to spin up the drive motor, and the time taken to move the head to a new position and wait for it to settle in place.

When the drive motor is started up or the a seek is requested, there will be a delay until the drive is ready for the next command. An interrupt is issued by the hardware when it is ready for the next command, and this will tell you that the drive is ready for your command.

In a single tasked environment (such as DOS), the only option is to have your driver constantly wait for an interrupt, and then respond to it. However, in a multi-tasked or multi-threaded environment, it is perfectly acceptable to write a driver which allows other tasks to be executed while waiting for the interrupt.

When performing a read or write operation, data may be transferred a byte at a time by reading from or writing to the appropriate port, or a sector/track at a time through the use of DMA channel 2. Programming the DMA controller is beyond the scope of this document, but will be described in a future document to be added to this site.

The registers, what they do and what commands can be used are all detailed in the

following sections. If there are any errors, I will gratefully accept any feedback you might like to send to me via e-mail .

Configuration of an FDC on a PC

The Floppy Controller on a PC uses a standard configuration. On the XT there are 3 ports available for control and data access registers. On the AT, there are 4, and on the PS/2 there are 6.

The base port address used for the controller is dependant on whether the controller is configured as the primary or secondary controller. This base address controls the port addresses used for each of the registers on the controller. It can additionally be noted that all floppy controllers on a PC use DMA channel 2 for data transfer during a read or write, and they all issue a hardware interrupt via IRQ6 to be serviced by INT 0eh by default.

?	**Primary Address**	**Secondary Address**	**Write (W) Read (R)**
?	?	?	?
base address	3f0h	370h	?
status register A (PS/2)	3f0h	370h	R
status register B (PS/2)	3f1h	371h	R
digital output register DOR	3f2h	372h	W
main status register	3f4h	374h	R
data rate select register (DSR)(PS/2)	3f4h	374h	W
data register	3f5h	375h	R/W
digital input register DIR (AT)	3f7h	377h	R
configuration control register (AT)	3f7h	377h	W
DMA channel	2	2	?
IRQ	6	6	?
INTR	0eh	0eh	?

Note that the controller can be configured differently from the defaults for handling interrupts.

FDC Registers

This section gives more detailed information on the use of the registers listed in the above table. Additionally, the first registers to be described are those common to each system described, and these will be followed by descriptions of AT specific registers and PS/2 specific registers.

Common Registers:

1. Digital Output Register DOR
2. Main Status Register
3. Data Register

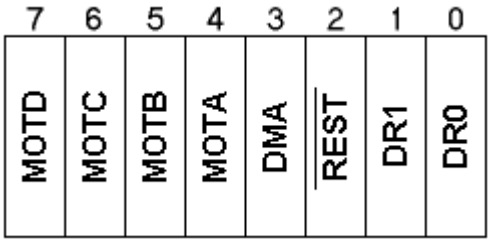
AT Specific Registers:

1. Digital Input Register DIR
2. Configuration Control Register

PS/2 Specific Registers:

1. Status Register A
2. Status Register B
3. Data Rate Select Register

Digital Output Register DOR



MOTD, MOTC, MOTB, MOTA:	**Motor control for floppy drive D, C, B, A**			
	1 = Start motor		0 = Stop motor	
DMA:	**DMA and IRQ channel**			
	1 = Enabled		0 = Disabled	
$\overline{\text{REST}}$:	**Controller reset**			
	1 = Controller enabled		0 = Execute controller reset	
DR1,DR0:	**Drive select**			
	00 = drive 0 (A)	01 = drive 1 (B)	10 = drive 2 (C)	11 = drive 3 (D)

This register is write only, and controls the drive motors, as well as selecting a drive and the DMA/IRQ mode, and resetting the controller.

If the REST bit is set, the controller is enabled, in order to accept and execute commands. If it is equal to 0, the controller ignores all commands and carries out an internal reset of all internal registers (except the DOR).

Note that a drive cannot be selected unless its motor is on, although setting the bits at the same time is acceptable.

Note also that drives 2 and 3 (floppy drives C and D) are not supported in all systems.

Example:
To start the motor on drive A and select it, ready for another operation, you would use the following:

MOTA = 1 (start motor for drive A)
DMA = 1 (assuming you want to use DMA and interrupts)
REST = 1 (Controller enabled, otherwise no other commands will be executed)
DR1,DR0 = 00 (Select drive A)
All other bits are set to 0

Thus 16 + 8 + 4 + 0 = 28, or 01Ch, is sent to port 3f2h (Drive A is usually on the primary controller).

In assembly language, this would be written as:

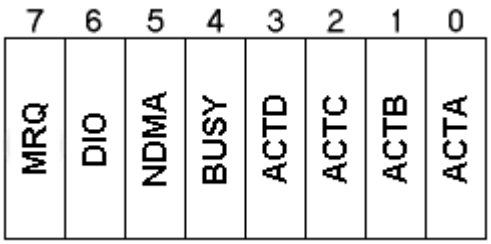
```
mov     al,01ch
mov     dx,03f2h
out     dx,al
```

Example:
To reset the controller, send 0 to port 3f2h. This turns off all motors, selects no drives (because drive A's motor is not active, it cannot be selected), disables the DMA and IRQ line, and resets the controller.

The code for this is as follows:

```
mov     al,0
mov     dx,03f2h
out     dx,al
```

Main Status Register



MRQ:	**main request**	
	1 = data register ready	0 = data register not ready
DIO:	**data input/output**	
	1 = controller ? CPU	0 = CPU ? controller
NDMA:	**non-DMA mode**	
	1 = controller not in DMA mode	0 = controller in DMA mode
BUSY:	**instruction (device busy)**	
	1 = active	0 = not active
ACTD, ACTC, ACTB, ACTA:	**drive D, C, B, A in positioning mode**	
	1 = active	0 = not active

The MSR is read-only, and contains the controller's status information. This register can be read whatever else the controller is doing.

It should be noted that this register is different from status registers ST0-ST3, which contain data concerning the last command executed. These registers are accessed via the data registers.

Bit 7 (MRQ) indicates whether the controller is ready to receive or send data or commands via the data register.

DIO is used to provide an indication of whether the controller is expecting to receive data from the CPU, or if it wants to output data to the CPU.

If the controller is set up to use DMA channel 2 to transfer data to or from main memory, the NDMA bit is not set. If this bit is set, data transfer is carried out exclusively by means of read or write commands to the data register. In this case, the controller issues a hardware interrupt every time that it either expects to receive or wants to supply a data byte.

Bit 4 indicates whether the controller is busy or not. If the bit is set, the controller is currently executing a command.

Bits 0-3 indicate which (if any) drive is currently in the process of positioning it's read/write heads, or being recalibrated.

Note that the delay waiting for the controller to be ready for a read or write can be as much as 175µs on an Intel controller, and longer on older controllers.

Example:
To test whether the controller is ready to receive commands and data, it is necessary to test MRQ and DIO. This involves reading the port, masking the bits, and doing a comparison on the result (to test the values of the bits).

In assembly language, this is can be written as:

```
mrqloop:
    mov dx,03f4h
    in  al,dx
    and al,0c0h
    cmp al,080h
    jne mrqloop
```

This code will keep looping until the controller says that it is ready to receive data. Note that if the controller is expecting to output data to the CPU, this code will not spot that. You would need to add another couple of lines of code if you need to check for both possibilities (In general, you would know from previous commands whether the controller should be expecting or offering data, and so only need to check for one condition).

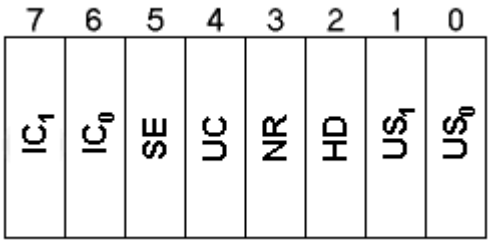
Data Register

The data register is an 8 bit register, like each of the other registers, which provides indirect access to a stack of registers. A command can be one to nine bytes in length, and the first byte tells the controller how many more bytes to expect. The controller sends the command bytes to the correct registers in it's stack, saving the programmer from the need to use a separate index register, as is the case in some other devices (e.g. some VGA registers).

Some controllers, such as the i82077A, have a buffer, with a programmable threshold, allowing the data to be transferred several bytes at a time. This helps to speed up the transfer of data and commands, as well as reducing the response time seen on the µPD765.

Following some of the commands, the values in the status registers are returned. The layout of the status registers follows, with the commands being listed at the end of this document.

Status Register ST0



IC_1,IC_0: interrupt code

00??/font> normal termination;
command terminated without any errors

01??/font> abnormal termination;

the controller started execution of the command, but couldn't terminate it correctly

10??/font> invalid command;
the controller could not start command execution

11??/font> abnormal termination by polling;
drive became not ready

SE: **seek end**

The controller has completed a seek or a calibration command,
or has correctly executed a read or write command which has an implicit seek

UC: **unit check**

Set if the drive faults or if a recalibrate cannot find track 0 after 79 pulses.

NR: **drive not ready**

HD: **head currently active**

1??/font> head 1

0??/font> head 0

US $_1$, US $_0$: currently selected drive (unit select)

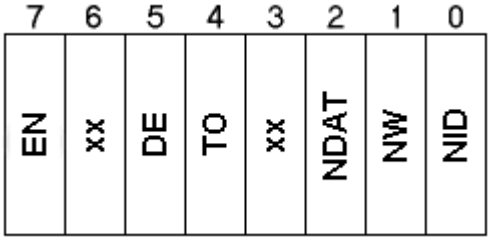
00??/font> drive 0 (A:)

01??/font> drive 1 (B:)

10??/font> drive 2 (C:)

11??/font> drive 3 (D:)

Status Register ST1



EN: **End of Cylinder**

Set when the sector count exceeds the number of sectors on a track.
i.e. the controller attempts to access a sector after the last sector of a cylinder.

xx: **bit unused**

value always equal to 0

DE: **data error**

Set if the controller detected an error in the ID address field or the data field of a sector

TO: **time-out**

Set for a data overrun;
No signal received from the DMA controller or CPU within the required time period.

NDAT: **no data**

Set爛f:?/font> The addressed sector in a *read sector* or *read deleted sector* cannot be found by the controller.

OR: ?/font> The controller cannot read the ID address mark in response to a *read ID* command without error.

OR:?/font> The controller cannot correctly determine the sequence of sectors in a *read track* command

NW: **not writable**

Set if the disk in the selected drive is write protected while the controller attempts to execute a write command.

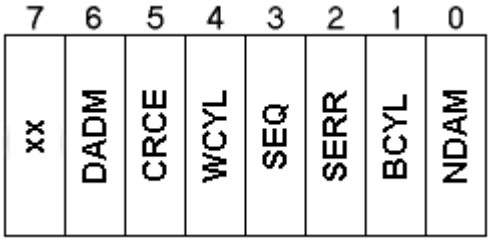
NID: **no address mark**

Set爛f:?/font> The ID address mark was not found after one complete disk revolution

OR:?/font> The controller could not find:

a data address mark DAM
a deleted data address mark DAM
on the specified track.

Status Register ST2



xx: **bit unused**

value always equal to 0

DADM: **deleted address mark**

Set if:?/font> A deleted data address mark DAM is detected when a *read sector* command is being executed.

OR:?/font> A valid data address mark DAM is detected when a *read deleted sector* command is being executed.

CRCE: **CRC error in data field**

Set if a CRC error was detected in the data field of the sector.

WCYL: **wrong cylinder**

Set if the track address in the controller and the track address in the ID address mark are different.

SEQ: **seek equal**

Set爛f: controller is a µPD765 and the condition *seek equal* is fulfilled

else: SGL is not used, this field is always equal to 0

SERR: **seek error**

Set爛f: controller is a µPD765 and the controller did not find the corresponding sector when seeking on the cylinder.

else: SERR is not used, this field is always equal to 0

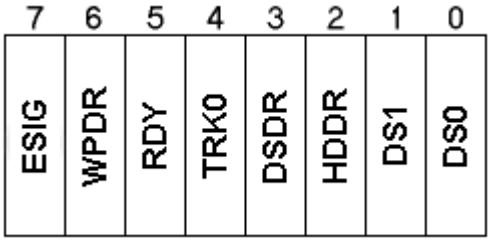
BCYL: **bad cylinder**

This field indicates that the track address in the ID address mark differs from the track address in the controller.
The value equals **ffh** , indicating a bad track with a physical error, according to the IBM soft sector format.

NDAM: **not data address mark DAM**

Set if the controller cannot find a valid or deleted data address mark DAM.

Status Register ST3



ESIG: **error**

Set爛f: controller is a µPD765 and the drives error signal is active i.e. an error has occurred

else: ESIG is not used and is always equal to 0

WPDR: **write protection**

Set if the disk is write protected (indicates the write-protection line is active).

RDY: **ready**

Set爛f: controller is a µPD765 and the drive is ready (indicates the ready signal of the drive is active).

else: RDY is not used and this field is always set

TRK0: **track 0**

The head is above track 0 (the TRK0 signal of the drive is active).

DSDR: **double sided drive**

The drive is double sided (indicates the DSDR signal is active).

HDDR: **head**

This bit indicates the status of the HDSEL signal of the drive:
1 = head 1 active,

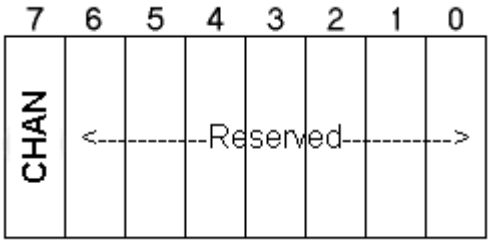
0 = head 0 active

DS1,燚S0: drive select

Both bits indicate the select signals DS1 and DS0 of the drive:
00 = drive 0 (A:)
01 = drive 1 (B:)
10 = drive 2 (C:)
11 = drive 3 (D:)

Digital Input Register

DIR (AT)



CHAN: **Disk Change**

1 = disk changed since last command 0 = disk not changed

RAT1,燻AT2: **Data Rate**

00 = 500 kbits/s
01 = 300 kbits/s
10 = 250 kbits/s
11 = 1Mbits/s

$\overline{\text{HiDe}}$ **High-density Rate**

1 = data rate 250kbits/s or 300 kbits/s
0 = data rate 1Mbits/s or 500 kbits/s

DMA: **Value of DMA bit in DOR**

NOPR: **Value of NOPR bit in Control Configuration Register**

This register is available only in the AT and the PS/2, and is read only. The above diagrams show that there are 3 different configurations for this register, differing in the AT, PS/2 and PS/2 Model 30. These registers all allow you to detect a disk change by reading bit 7, with additional information available for the PS/2 and Model 30 variants.

For each of the registers, when bit 7 is set, it indicates that the disk has been changed since the last command was executed. This can be used by a driver to speed up access to data, by buffering of data. When buffering data, if the disk has not been changed, required sectors can already be in memory when requested (for example, a whole track can be read at a time and stored in a buffer, including the possibility of storing multiple tracks when sectors are read from different cylinders). If this happens, the data only needs reading from disk when a new cylinder is being read from, or when the disk has been changed.

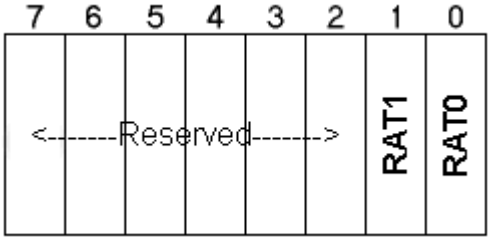
In the PS/2 and Model 30, the registers also contain information about the current data transfer rate. For Model 30, this information is read from bits 1 and 0, while in the other PS/2 models, the data is read from bits 2 and 1. This data can be used when the rate is set through the Control Configuration Register to check that the rate has been correctly set, or to check what rate the controller is set to at any time.

In PS/2 models other than Model 30, bit 1 can be read to help determine whether the controller is set to a high or low data transfer rate for a high density disk.

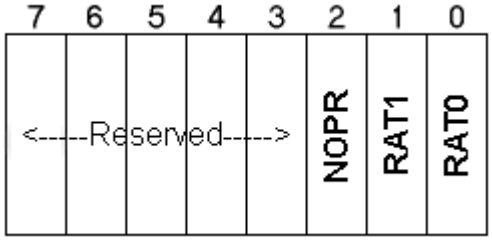
In Model 30, bit 3 corresponds to bit 3 of the DOR. bit 2 corresponds to bit 2 of the CCR (only used in this model). In both these cases, the value is read-only in the DIR, and can be written to in the corresponding register.

Control Configuration Register

CCR (AT and PS/2)



CCR (PS/2 Model 30)



RAT1,嫌AT0: **Data Transfer Rate**

00 = 500kbits/s
01 = 300kbits/s
10 = 250kbits/s
11 - 1Mbits/s

NOPR: **(No) Precompensate**

0 = Precompensation Enabled (standard)

1 = No Precompensation

Data Transfer Rates

Rate	Disk Capacity	Size	Drive Capacity

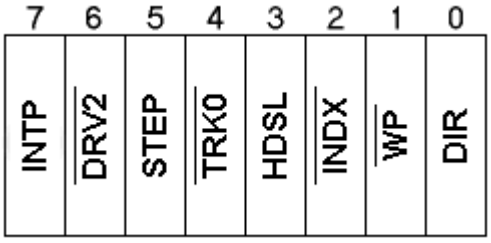
250 kbits/s	360燾byte	5¼"	360燾byte
	720燾byte	3½"	1.44燤bytes
	720燾byte	3½"	720燾byte
300 kbits/s	720燾byte	3½"	720燾byte
	360燾byte	5¼"	1.2牋Mbyte
500 kbits/s	1.2牋Mbyte	5¼"	1.2牋Mbyte
	1.44燤bytes	3½"	1.44燤bytes

In the AT and all PS/2 models, the data transfer rate can be set through bits 1 and 0 of the CCR. Valid transfer rates are shown in the table immediately above, and the table below the diagram shows what values need to be sent to these two fields to set the appropriate rate.

In the Model 30 it is also possible to program the precompensation through bit 2 of this register. The default is for precompensate to be enabled, with this field set to 0. Setting the field to 1 turns precompensation off.

Status Registers A and B

Status Register A (PS/2)



INTP: Interrupt Pending
0 = interrupt signal inactive
1 = active

DRV2 : 0 = 2 drives connected
1 = one drive only

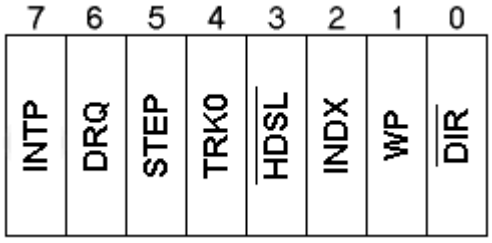
STEP: Stepper Pulse
0 = no pulse
1 = pulse is submitted

TRK0 : Track 0
0 = head not above track 0
1 = head above track 0

HDSL: Head Select
0 = head 0
1 = head 1

INDX : Index Mark
0 = detected

Status Register A (Model 30)



INTP: Interrupt Pending
0 = interrupt signal inactive
1 = active

DRQ: DMA Request
0 = not active
1 = active

STEP: Step Pulse
0 = pulse is submitted
1 = no pulse

TRK0: Track 0
0 = head above track 0
1 = head not above track 0

HDSL : Head Select
0 = head 1
1 = head 0

INDX: Index Mark

1 = not detected

$\overline{WP}$: Write Protection

0 = disk write protected
1 = not write protected

DIR: Direction of Head

0 = outwards (to smaller cylinder numbers)

1 = inwards (to higher cylinder numbers)

0 = not detected
1 = detected

WP: Write Protection

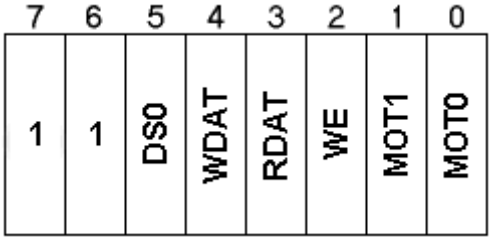
0 = disk not write protected
1 = disk write protected

$\overline{DIR}$: Head Direction

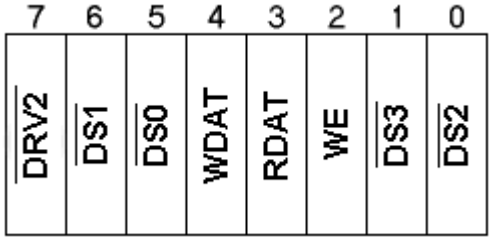
0 = inwards (to higher cylinder numbers)

1 = outwards (to lower cylinder numbers)

Status Register B (PS/2)



Status Register B (Model 30)



DS0: Drive Select

0 = drive other than 0
1 = drive 0

WDAT: Write Data

0 = no data can be written to drive
1 = data can be transferred to drive

RDAT: Read Data

0 = no data can be read from drive
1 = data can be transferred from drive

WE: Write Enabled

0 = head is set to read data
1 = head is activated for data writes

MOT1,燗OT0: Motor of drive 1, 0

0 = motor is switched off
1 = motor is switched on

$\overline{DRV2}$: 0 = two drives connected
1 = only one drive connected

$\overline{DS1}$: Drive Select 1

0 = drive 1 selected
1 = drive not selected

$\overline{DS0}$: Drive Select 0

0 = drive 0 selected
1 = drive not selected

WDAT: Write Data

0 = no data can be written
1 = data can be transferred to the drive

RDAT: Read Data

0? data cannot be read from drive
1 = data can be read from drive

WE: Write Enabled

0 = head enabled to read data
1 = head enabled to write data

$\overline{DS3}$: Drive Select 3

0 = drive 3 selected
1 = drive not selected

$\overline{DS2}$: Drive Select 2

0 = drive 2 selected
1 = drive not selected

These registers are read only, and only available on the PS/2.

By the use of the two status registers A and B on a PS/2, it is possible to read the status of

the control lines between the floppy controller and drive. Register bits $\overline{\text{DRV2}}$, $\overline{\text{TRK0}}$, $\overline{\text{INDX}}$, $\overline{\text{WP}}$ and RDAT indicate the status of the corresponding data lines. Note that the bit values vary between Status Registers A and B on Model 30 and other PS/2 models. Some of the values are also detectable through other registers and the Status Registers readable through the data register on the AT.

Data Rate Select Register

If you know of any source of information on this register (available only on the PS/2), please e-mail me with details. As soon as I am able to find the information, it will be added to this page.

Command Set

There are a total of 13 commands available on the µPD765 and compatible FDCs. A further 4 commands are available on the 8207x controllers.

The *sector identification* consists of the cylinder, head, sector number and sector size. This tells the controller the position and number of sectors to perform this command on.

All commands and status bytes are transferred via the data register, at port 37fh or 377h. Before the command can be written or the status byte read, it is necessary to read the MRQ bit in the main status register. This determines whether the data register is ready to supply or receive a byte. It is also necessary to fix the drive format prior to any read, write or format operation.

For most data transfers, DMA is used. Although the programming of the DMA is beyond the scope of this document, I have provided some example code showing how to set up the DMA for writing a single sector from a floppy disk drive to main memory. All data transfers concern all sectors from the start sector to the end of the track. The operation can be stopped earlier by either setting the command byte *track length/max. sector number* to a value which indicates the last sector to be operated on, or setting the count value of the DMA controller so that it issues a TC (Terminal Count) signal after the required number of sectors are transfered (the latter method is demonstrated in the example DMA code below).

If you want a command to be executed on both heads, you need to set the Multiple Track Bit. This tells the controller to operate on the programmed head first, then to carry out the same command from the start of the other head.

Once the command has been completed, the status registers ST0-3 return information which can help you either confirm correct execution of the command, or determine the cause of an error.

The commands fall into 3 categories. Data transfer commands and control commands are available on all controllers. The extended commands are only available on the AT or PS/2.

List of Valid Commands

Data Transfer Commands

Command Name	Function
read complete track	x2h
write sector	x5h
read sector	x6h
write deleted sector	x9h
read deleted sector	xch
format track	xdh

Control Commands

Command Name	Function
fix drive data	x3h
check drive status	x4h
calibrate drive	x7h
check interrupt status	x8h
read sector ID	xah
seek/park head	xfh
invalid command	all invalid opcodes

Extended Commands

Command Name	Function
register summary	0fh
determine controller version	x10h
verify	x16h
seek relative	1xfh

In each of the above, 'x' refers to bits 7-5 of byte 0 of the command. Note that in the *seek relative* command, x refers to just bits 6 and 5, as bit 7 is always set to 1. The remainder of the function number refers to bits 4-0. Bit 4 is only used on the AT and PS/2, in 2 of the extended commands.

The function number given is the first byte of the command it applies to. The commands vary in size from 1 to 9 bytes, and the controller knows from the function number how many more bytes to expect. For example, if the first byte received by the data register is **66h** , in which the upper 5 bits are 06h, the controller knows that you are sending a *read sector* command, and it expects 8 more bytes from the CPU. Additionally, you do nto need to worry about which of the internal registers each byte of any command has to be directed to, as the controller does that for you.

Fields common to many of the commands include the following:

M: **Multi-track operation**
1 = carry out operation on both tracks of programmed cylinder.
0 = carry out operation on single track.

F: **FM/MFM mode**
1 = operate in MFM (double density) mode (default)

	0 = operate in FM (single density) mode
S:	**Skip mode** 1 = skip deleted data address marks, 0 = do not skip
HD:	**head number** (always equal to head address in byte 3 of all commands using a sector ID)
DR1,DR0:	**Drive:** 00 = drive 0 (A); 01 = drive 1 (B); 10 = drive 2 (C); 11 = drive 3 (D)
Cylinder, Head, Sector Number:	**Address of first sector to read.**
Sector size code:	**Sector size** = 128 * 2^x where x is the value in this field. eg If this field is 2 (the default), sector size = 128 * 2^2 = 512 bytes
Track length/Max sector number:	**Number of sectors per track or max. sector number** to operate command on.
Length of GAP 3:	Standard value = 42, minimal value = 32 (5¼") standard value = 27 (3½")
Data length:	**length of data to read** in bytes (only valid if sector size = 0, else equal 0ffh)

Data Transfer Commands

These are the commands used to transfer data between a disk and main memory, or to format a track.

Each of these commands returns its' results in the same format, which is only described for the *read track* command.

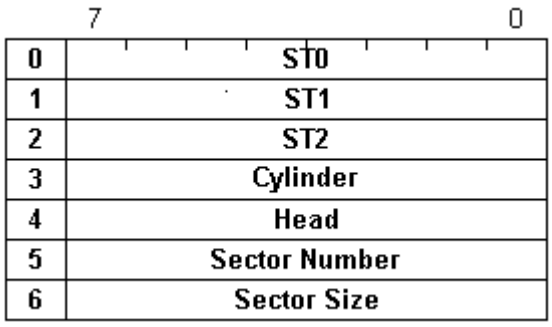
Read Track (x2h)

Bit Byte	7	6	5	4	3	2	1	0
0	0	F	S	0	0	0	1	0
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Head							
4	Sector Number							
5	Sector Size							
6	Track Length							
7	Length of GAP3							
8	Data Length							

When a *read track* command is issued, the data of a single complete track is read. The sector specification in the command phase is ignored for this command, and the reading starts with the first sector after the index address mark **IDAM** , reading sector by sector (paying no attention to the logical sector number given in the ID address mark), until the end of the track is reached.

The track is treated as a contigusous data block, and the read buffer in main memory should be large enough to store this amount of data. Also, multi-track operations are not allowed with this command. If you want to read the same track on both heads, you have to send the command twice, unlike other read commands which allow for multi-track operations with a single command.

Results Phase



	7 ... 0
0	ST0
1	ST1
2	ST2
3	Cylinder
4	Head
5	Sector Number
6	Sector Size

ST0,燬T1,燬T2: Status Registers 0 to 2

Cylinder,燽ead, sector number,燭ector size: Sector ID. (see table below)

Sector ID in the Result Phase

M牋	HD $_{prog}$ 牋	Last sector affected牋 by command	Cylinder牋	Head牋	Sector牋	Sector Size
0	0	before end of track	cyl $_{prog}$	HD $_{prog}$	sec $_{prog}$	siz $_{prog}$
0	0	end of track	cyl $_{prog+1}$	HD $_{prog}$	1	siz $_{prog}$
0	1	before end of track	cyl $_{prog}$	HD $_{prog}$	sec $_{prog}$	siz $_{prog}$
0	1	end of track	cyl $_{prog+1}$	HD $_{prog}$	1	siz $_{prog}$
1	0	before end of track	cyl $_{prog}$	HD $_{prog}$	sec $_{prog}$	siz $_{prog}$
1	0	end of track	cyl $_{prog}$	HD $_{prog}$	1	siz $_{prog}$
1	1	before end of track	cyl $_{prog}$	(inverse of) HD $_{prog}$	sec $_{prog}$	siz $_{prog}$
1	1	end of track	cyl $_{prog+1}$	(inverse of) HD $_{prog}$	1	siz $_{prog}$

HD $_{prog}$: programmed head牋 sec $_{prog}$: programmed sector

cyl $_{prog}$: programmed cylinder牋 siz $_{prog}$: programmed sector size

The Sector ID consists of Cylinder, Head, Sector and Sector Size. Given the values programmed for M (multi-track) and HD (head number), the values for the sector ID in the results phase indicate whether the last sector affected was the last sector of the command or not.

Write Sector (x5h)

Bit Byte	7	6	5	4	3	2	1	0
0	M	F	0	0	0	1	0	1
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Head							
4	Sector Number							
5	Sector Size							
6	Track Length/Max. Sector Number							
7	Length of GAP3							
8	Data Length							

The *write sector* command transfers one or more sectors from main memory to the controller, from where it is transferred to the disk. As the controller writes each sector, it also writes a valid *data address mark* to the disk. This command can operate on both heads, starting from the first sector of the second head after reaching the end of the first head.

The results phase is the same as for the read track command.

Read Sector (x6h)

Bit Byte	7	6	5	4	3	2	1	0
0	M	F	S	0	0	1	1	0
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Head							
4	Sector Number							
5	Sector Size							
6	Track Length/Max. Sector Number							
7	Length of GAP3							
8	Data Length							

The *read sector* command reads one or more sectors with a valid data address mark from the disk, and transfers the data into main memory. This command can operate on both heads.

The results phase is the same as for the read track command.

Write Deleted Sector (x9h)

Bit Byte	7	6	5	4	3	2	1	0
0	M	F	0	0	1	0	0	1
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Head							
4	Sector Number							
5	Sector Size							
6	Track Length/Max. Sector Number							
7	Length of GAP3							
8	Data Length							

The *write deleted sector* command is the same as the *write sector* command, except that for each sector written a *deleted data address mark* is written instead of the normal data address mark.

The results phase is the same as for the read track command.

Read Deleted Sector (xch)

Bit Byte	7	6	5	4	3	2	1	0
0	M	F	0	0	1	1	0	0
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Head							
4	Sector Number							
5	Sector Size							
6	Track Length/Max. Sector Number							
7	Length of GAP3							
8	Data Length							

The *read deleted sector* command is the same as the *read sector* command, except that only sectors with a deleted data address mark can be read. All sectors with valid data address marks will be ignored.

The results phase is the same as for the read track command.

Format Track (xdh)

Bit Byte	7	6	5	4	3	2	1	0
0	0	F	0	0	1	1	0	1
1	x	x	x	x	x	HD	DR1	DR0
2	Sector Size							
3	Track Length							
4	Length of GAP3							
5	Fill Byte							

This command formats a single track. A 4 byte format buffer must be provided for each sector of the track. The buffer holds the sector ID of the corresponding sector. The format buffer should be large enough to hold the data required for all sectors of the track. The buffer format is shown below.

For ease of use, the DMA controller should be programmed to enable the controller to read the format buffer via DMA channel 2. Alternatively, the format data can be transferred by the use of interrupt-driven data exchange. The controller issues a hardware interrupt before fromatting each sector. The handler can then transfer the 4 byte format information for the next sector to be formatted.

The formatting begins once the drive has provided a signal on the IDX line, indicating the beginning of the track. Sectors are formatted continuously until the drive passes the same signal again, this time indicating the end of the track (note that the start and end of the track are at the same point, marked by an index hole on the disk).

For the format command, the length of the GAP field is larger than when reading or writing data. Unless i find any further information on the GAP length, I can only include the default GAP length in this document.

The results phase is the same as for the read track command.

Format buffer for one sector

Byte	Command
0	Track
1	Head
2	Sector Number
3	Sector Size

In the above table, *Sector Size* uses the same codes as for each of the commands described above.

Control Commands

This set of commands includes various miscellaneous commands relating to the status of a disk or drive, including seeking to a new cylinder and responding to invalid commands.

Fix Drive Data (x3h)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1
1	Step Rate				Head Unload Time			
2	Head Load Time							NDM

NDM: **Non-DMA Mode**
0 = Data Transfer via DMA
1 = Data Transfer not via DMA

Step Rate [ms]

Entry	Data Transfer Rate			
	1M	500k	300k	250k
0h	8.0	16	26.7	32
1h	7.5	15	25.0	30
2h	7.0	14	23.3	28
...	...	...	...	...
eh	1.0	2	3.3	4
fh	0.5	1	1.7	2

Head Unload Time [ms]

Entry	Data Transfer Rate			
	1M	500k	300k	250k
0h	128	256	426	512
1h	8	16	26.7	32
2h	16	32	53.3	64
...	...	...	...	...
eh	112	224	373	448
fh	120	240	400	480

Head Load Time

Entry	Data Transfer R		
	1M	500k	300k
0h	128	256	426
1h	1	2	3.3
2h	2	4	6.6
...	...	...	...
7eh	126	252	420
7fh	127	254	423

[ms]

This command is used to pass mechanical control data to the controller for the connected drives. It should be noted, as shown in the bottom 3 diagrams, that the values are also dependent on the data transfer rate, which in the AT and PS/2 is set in the control configuration register .

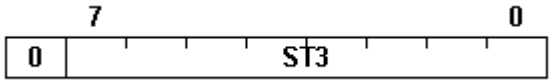
This command doesn't have a result phase.

Check Drive Status (x4h)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0
1	x	x	x	x	x	HD	DR1	DR0

This command provides status information relating to the state of the connected drives.

Result Phase



Status Register 3 contains drive information.

Calibrate Drive (x7h)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1
1	x	x	x	x	x	0	DR1	DR0

This command is used to position the read/write head to cylinder 0. If a seek error occurs in the course of a sector access, the head can be moved to an absolute cylinder to recalibrate the drive.

This command doesn't return a result phase, but after cmopletion an interrupt is issued. To check the status information of this command, you should issue a *check interrupt status* cmomand to determine the commands status information.

When the controller sees this command, it sets the DIR signal to 0, and passes the drive up to 79 step pulses. After each of these pulses, the controller checks the TRK0 signal. If it is active (htat is, the head is on track 0), the controller sets the SE bit in Status Register 0, and aborts the command. If TRK0 is not active after 79 step pulses, the controller sets bits SE and EC in Status Register 0, and terminates the command.

To calibrate the drive, you may have to issue several calibration commands, especially if the drive being calibrated has more than 80 tracks. After completion of the command, you should always check whether the head is correctly positioned over track 0, using the *check interrupt status* command. A calibration is always necessary after a power up, to initialise the head position correctly.

Check Interrupt Status (x8h)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0

This command is used to check status infromation about the state of the controller in the result phase when the controller has returned an interrupt.

The interrupt signal is reset by this command, which also determines the source of the interrupt via status register ST0. If the command is issued with no interrupts pending, a value of 80h is returned in ST0, corresponding to the message *invalid command* .

Interrupts are issued in the following cases:

At the beginning of the result phase of the commands:
- read sector
- read deleted sector
- write sector
- write deleted sector
- read track
- format track
- read sector ID
- verify

After completion of the following commands without a result phase:
- calibrate drive
- seek
- seek relative

For data exchange between main memory and controller when interrupt-driven data exchange is active and the controller is not using DMA.

Result Phase

	7 ... 0
0	ST0
1	Current Cylinder

Read Sector ID (xah)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	F	0	0	1	0	1	0
1	x	x	x	x	x	HD	DR1	DR0

This command is used to read the Sector ID of the first ID address mark the controller is able to detect. Using this command, it is possible to determine the current position of the read/write head. If no ID address mark can be read in one complete disk revolution, the controller issues an error message, detected from the values returned in ST0-2.

Result Phase

	7 ... 0
0	ST0
1	ST1
2	ST2
3	Cylinder
4	Head
5	Sector Number
6	Sector Size

The values in the sector ID are calculated in the same way as for the result phase of the read track command.

Seek/Park Head (xfh)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	1
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							

The Seek Head command, sometimes called the Park Head command, moves the read/write head to the specified cylinder. When the controller receives this command, it compares the programmed cylinder number and the current cylinder number. The direction signal (DIR) is set, and step pulses are issued until the two cylinder numbers match.

This command has no result phase. To verify successful completion of the command, it is necessary to check the head position immediately after completion of the command, using the check interrupt status command.

---

invalid command (all invalid opcodes)

Bit Byte	7	6	5	4	3	2	1	0
0	Invalid Opcode							

Whenever an invalid opcode is detected, the controller switches to a standby state, and bit 7 of ST0 is set. The same happens if *check interrupt status* is issued with no interrupts pending.

Result Phase

	7	0
0	ST0	

Extended Commands

These commands are not available on all controllers, being introduced in the AT and PS/2. If the controller does not support any of these commands, it will treat them as invalid commands.

Register Summary (0fh)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	1

This command returns the values read from the internal controller registers.

Result Phase

	7							0
0	Current Cylinder DR0							
1	Current Cylinder DR1							
2	Current Cylinder DR2							
3	Current Cylinder DR3							
4	Step Rate				Head Unload Time			
5	Head Load Time							NDM
6	Number of Sectors/Track Length							

Current烫ylinder燚R0,燚R1,燚R2燚R3:	Cylinder on drive 0, 1, 2, 3 where read/write head is currently positioned.
step爐ime, head爝nload爐ime, head爐oad爐ime:	mechanical characteristics set by the *fix drive data* command
NDM:	**non-DMA mode** 1 = DMA disabled 0 = DMA enabled
number of sectors/track length	**number of sectors per track**

Determine Controller Version (10h)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0

This command determines whether there is a controller present which supports the extended commands. If the controller does not support the extended commands, this command is treated as an invalid opcode, and an error message is returned.

Result Phase

	7							0
0	1	0	0	1	0	0	0	0

This result is only returned if an extended controller is installed.

Verify (x16h)

Bit Byte	7	6	5	4	3	2	1	0
0	M	F	S	1	0	1	1	0
1	EC	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Head							
4	Sector Number							
5	Sector Size							
6	Track Length/Max. Sector Number							
7	Length of GAP3							
8	Data Length/Verify Sectors							

EC: **enable count value**
1 = command byte 8 specifies the number of sectors to verify
0 = command byte 8 specifies the data length, if sector size = 0

data燴ength/verify燼ectors:牋 **If EC = 0 and sector size = 0:**
牋牋length of data to verify, in bytes.
Else:
牋牋number of sectors to verify.

This command is similar to the read command, except that it doesn't transfer data to main memory. One or more sectros with valid DAMs are read from the disk, and their CRC is calculated. This value is compared to the read CRC, inorder to check the internal consistency of the data. As no data is transferred, the command cannot be aborted by a TC signal from the DMA controller. However, if youset the EC bit to 1, the controller issues an implicit TC signal when the count value in *data length/verify sectors* is decremented to 0. In that case, *data length/verify sectors* indicates the number of sectors to be verified. A value of 0 in this byte tells the controller to check 256 sectors. When EC is set to 0, *data length/verify data* should be set to ffh.

Result Phase

	7 ... 0
0	ST0
1	ST1
2	ST2
3	Cylinder
4	Head
5	Sector Number
6	Sector Size

Seek Relative (1xfh)

Bit Byte	7	6	5	4	3	2	1	0
0	1	DIR	0	0	1	1	1	1
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder Step							

DIR: **Step Direction**
1 = inward (to larger cylinder numbers)
0 = outward (to smaller cylinder numbers)

cylinder step: **number of cylinders to step**

This command is sued to move the read/write head relative to the current cylinder. There is no result phase for this command, but its' result can be checked through the use of either the *read sector ID* or the *register dump* command.

Sample code

Initialise DMA for writing a single sector from FDC to main memory

Please note that this code was added in a hurry and has not yet been tested. I wanted to publish this page with DMA initialisation code before going on holiday. The code will be tested and, if necessary, adjusted, late in June (1999), when I get back from my vacation.

```
;*****************************************************************************
; This code expects to receive the data buffer address in ES:BX
;
; ES:  Buffer segment
; BX:  Buffer offset
;
; The address is a 20 bit segmented address, calculated from Segment*16 + Offset
;
; Bits 19-16 of the address form the entry for the DMA page entry
; Bits 15-8 of the address form the high-order byte for the DMA address register
; Bits  7-0 of the address form the low-order byte for the DMA address register
;
; For more information on this, it will be necessary to read other literature
;    specifically targetting the DMA controller.
;*****************************************************************************

disable_dma1:          ; disable DMA 1

mov     al, 14h    ; output 14h to the command register to disable and
out     08h, al    ; initialise the DMA controller

mode:                  ; set up DMA transfer mode for channel 2
mov     al, 56h    ; set  up for a single write transfer to main memory
out     0bh, al    ; using DMA channel 2
; for a read, output 5ah
```

```
get_address:           ; get the buffer address, and split into component parts
; as required by the DMA controller.
mov     ax, es     ; load buffer segment into AX
mov     cl, 04h    ;
shl     ax, cl     ; shift value in ax left 4 times
add     ax, bx     ; add offest + buffer. AX now contains high and low
; for the DMA address register.
jc      carry      ; if carry is set, jump to carry

no_carry:
mov     bx, es     ; load segment of buffer into BX
mov     cl, 04h    ;
shr     bh, cl     ; shift right BH 4 times. BH now contains the value
; for the DMA page segment
jmp  buffer_address; output the buffer address

carry:
mov     bx, es     ; load segment of buffer into BX
mov     cl, 04h    ;
shr     bh, cl     ; shift right BH 4 times.  BH now contains the high
; bits of the segment
adc     bh, 00h

buffer_address:        ; output the address to the DMA controller
out     0ch, al    ; reset flip-flop. I am not sure what this does,
; but my reference shows this as necessary.
out     04h, al    ; output low order address byte to address register
mov     al, ah     ;
out     04h, al    ; output high-order address byte to address register
mov     al, bh     ;
out     81h, al    ; load page register with page value

count:                 ; set up count register
out     0ch, al    ; reset flip-flop
mov     al, 0ffh   ;
out     05h, al    ;
mov     al, 01h    ;
out     05h, al    ; load 511 into count register (to read one sector)
; it should be noted that for multiple transfers, the
; second value output to port 05h is equivalent to
; (2*number of sectors)-1

release_channel:
mov     al, 02h
out     0ah, al    ; release channel 2

enable_dma1:           ; enable DMA 1
mov     al, 10h
out     08h, al   ; this is the value for enabling the DMA for the mode required
```

Once I have written my own floppy disk drivers for the operating system I am working on, I'll add a link to that. Until then, I'm happy to continue replying to any questions I receive by email. Please note that I am a programmer, not a hardware expert, and as such may not be able to answer questions that relate to how to produce hardware using a FDC.

If you have any comments on the content of this page, including errors and typos, please contact me: